Lab 12 Solutions **lab12.zip (lab12.zip)**

Solution Files

This lab has several files. Remember to write in `lab12.scm` for the Scheme questions, `lab12.sql` for the SQL questions, and `lab12.py` for all other questions.

> **Note:**
>
> - Some problems have skeletons. You are encouraged to use them, but it is not required.
> - Solutions are released for this lab. We recommend that you try the lab on your own and then refer to solutions afterwards.

Required Questions

Linked Lists

Q1: Link Pop

Implement `link_pop`, which is similar to the `pop()` method for lists. `link_pop` takes in a Linked List `lnk` and optionally one extra argument `index`. If provided a second argument, `link_pop` will remove the value at `index` from the list and return it. If not given a second argument, `link_pop` will remove the last value from the list and return it. Assume that you will never be asked to remove an element at index 0 or at an index out of range.

> **Note:** In the function definition, `index=-1` denotes that when an `index` argument is not provided, the value of `index` defaults to `-1`.

```
def link_pop(lnk, index=-1):
    '''Implement the pop method for a Linked List.

    >>> lnk = Link(1, Link(2, Link(3, Link(4, Link(5)))))
    >>> removed = link_pop(lnk)
    >>> print(removed)
    5
    >>> print(lnk)
    <1 2 3 4>
    >>> link_pop(lnk, 2)
    3
    >>> print(lnk)
    <1 2 4>
    >>> link_pop(lnk)
    4
    >>> link_pop(lnk)
    2
    >>> print(lnk)
    <1>
    '''
    if index == -1:
        while lnk.rest.rest is not Link.empty:
            lnk = lnk.rest
        removed = lnk.rest.first
        lnk.rest = Link.empty
    else:
        while index > 1:
            lnk = lnk.rest
            index -= 1
        removed = lnk.rest.first
        lnk.rest = lnk.rest.rest
    return removed
```

Use Ok to test your code:

```
python3 ok -q link_pop
```

Trees

Q2: Prune Min

Write a function that prunes a `Tree t` mutatively. `t` and its branches always have zero or two branches. For the trees with two branches, reduce the number of branches from two to one by keeping the branch that has the smaller label value. Do nothing with trees with zero branches.

Prune the tree in a direction of your choosing (top down or bottom up). The result should be a linear tree.

```
def prune_min(t):
    """Prune the tree mutatively.

    >>> t1 = Tree(6)
    >>> prune_min(t1)
    >>> t1
    Tree(6)
    >>> t2 = Tree(6, [Tree(3), Tree(4)])
    >>> prune_min(t2)
    >>> t2
    Tree(6, [Tree(3)])
    >>> t3 = Tree(6, [Tree(3, [Tree(1), Tree(2)]), Tree(5, [Tree(3), Tree(4)])])
    >>> prune_min(t3)
    >>> t3
    Tree(6, [Tree(3, [Tree(1)])])
    """
    if t.branches == []:
        return
    prune_min(t.branches[0])
    prune_min(t.branches[1])
    if (t.branches[0].label > t.branches[1].label):
        t.branches.pop(0)
    else:
        t.branches.pop(1)
    return # return statement to block alternate from running
```

Use Ok to test your code:

```
python3 ok -q prune_min
```

Scheme Data Abstraction

Vehicles

Say we have an abstract data type for owners and vehicles.

1. The `owner` abstraction keeps track of the `name` of a vehicle-owner as well as the `age` they got their license.
2. The `vehicle` abstraction keeps track of the `model`, the `year`, and `previous-owners` of the vehicle

You can find the constructors for these classes below:

```
(define (make-owner name age) (cons name (cons age nil)))
(define (make-vehicle model year previous-owners) (cons model (cons year previous-owner
```

Q3: Vehicles: Selectors

Implement `get-model`, `get-year`, and `get-owners`. These functions take in a `vehicle` abstraction, and return the relevant attribute. For example, `get-model` takes `vehicle` as an input and returns the `model`.

> *Note*: `get-owners` returns a list of owner objects.

```
(define (get-model vehicle)
    (car vehicle)
)

(define (get-year vehicle)
    (car (cdr vehicle))
)

(define (get-owners vehicle)
    (cdr (cdr vehicle))
)
```

Use Ok to test your code:

```
python3 ok -q vehicles_selectors
```

Q4: Old

Implement `older-vehicle`. This method takes in two `vehicle`s as input and returns the model of the older one.

> Be sure to keep the abstraction barrier in mind! Feel free to use the selector functions we defined in the previous question.

```
(define (older-vehicle vehicle1 vehicle2)
    (define year1 (get-year vehicle1))
    (define year2 (get-year vehicle2))
    (define diff (- year1 year2))
    (if (> diff 0)
        (get-model vehicle2)
        (get-model vehicle1)
    )
)
```

Use Ok to test your code:

```
python3 ok -q vehicles_older
```

Q5: New Owner

Implement `new-owner`. This method takes in a `vehicle` and `owner` as input, and returns a *new* `vehicle` abstraction with the `previous-owners` list updated to reflect the new `owner` added.

> Be sure to keep the abstraction barrier in mind!

```
(define (new-owner vehicle owner)
    (define model (get-model vehicle))
    (define year (get-year vehicle))
    (define owners (cons owner (get-owners vehicle)))
    (make-vehicle model year owners)
)
```

Use Ok to test your code:

```
python3 ok -q vehicles_new_owner
```

Q6: Owner Names

Implement `owners-names`. This method takes in a `vehicle` as input and returns a list of only the names of each owner stored in the `previous-owners` list within the specified `vehicle`.

> **Hint:** The *map* function could be helpful here!

```
(define (owners-names vehicle)
    (map (lambda (owner) (get-name owner)) (get-owners vehicle))
)
```

Use Ok to test your code:

```
python3 ok -q vehicles_owners_names
```

SQL

Pizza

In each question below, you will define a new table based on the following tables. The first defines the names, opening, and closing hours of great pizza places in Berkeley. The second defines typical meal times (for college students). A pizza place is open for a meal if the meal time is at or within the `open` and `close` times.

Your tables should still perform correctly even if the values in these tables were to change. Don't just hard-code the output to each query.

Q7: Opening Times

You'd like to have lunch before 1pm. Create a `opening` table with the names of all Pizza places that open before 1pm, listed in reverse alphabetical order.

```
-- Pizza places that open before 1pm in alphabetical order
CREATE TABLE opening AS
  SELECT name FROM pizzas WHERE open < 13 ORDER BY name DESC;
```

Use Ok to test your code:

```
python3 ok -q opening
```

Q8: Double Pizza

If two meals are more than 6 hours apart, then there's nothing wrong with going to the same pizza place for both, right? Create a `double` table with three columns. The first columns is the earlier meal, the second is the later meal, and the third is the name of a

pizza place. Only include rows that describe two meals that are **more than 6 hours apart** and a pizza place that is open for both of the meals. The rows may appear in any order.

```
-- Two meals at the same place
CREATE TABLE double AS
  SELECT a.meal, b.meal, name
         FROM meals AS a, meals AS b, pizzas
         WHERE open <= a.time AND a.time <= close AND
               open <= b.time and b.time <= close AND
               b.time > a.time + 6;
```

```
-- Example:
SELECT * FROM double WHERE name="Sliver";
-- Expected output:
--   breakfast|dinner|Sliver
```

Use Ok to test your code:

```
python3 ok -q double
```

Recommended Questions

The following problems are not required for credit on this lab but may help you prepare for the final.

Iterators

Q9: Repeated

Implement `repeated`, which takes in an iterator `t` and returns the first value in `t` that appears `k` times in a row.

> **Note:** You can assume that the iterator `t` will have a value that appears at least `k` times in a row. If you are receiving a `StopIteration`, your `repeated` function is likely not identifying the correct value.

Your implementation should iterate through the items in a way such that if the same iterator is passed into `repeated` twice, it should continue in the second call at the point it left off in the first. An example of this behavior is in the doctests.

```
def repeated(t, k):
    """Return the first value in iterator T that appears K times in a row.
    Iterate through the items such that if the same iterator is passed into
    the function twice, it continues in the second call at the point it left
    off in the first.

    >>> s = iter([10, 9, 10, 9, 9, 10, 8, 8, 8, 7])
    >>> repeated(s, 2)
    9
    >>> s2 = iter([10, 9, 10, 9, 9, 10, 8, 8, 8, 7])
    >>> repeated(s2, 3)
    8
    >>> s = iter([3, 2, 2, 2, 1, 2, 1, 4, 4, 5, 5, 5])
    >>> repeated(s, 3)
    2
    >>> repeated(s, 3)
    5
    >>> s2 = iter([4, 1, 6, 6, 7, 7, 8, 8, 2, 2, 2, 5])
    >>> repeated(s2, 3)
    2
    """
    assert k > 1
    count = 1
    last_item = None
    while True:
        item = next(t)
        if item == last_item:
            count += 1
        else:
            last_item = item
            count = 1
        if count == k:
            return item
```

Use Ok to test your code:

```
python3 ok -q repeated
```

Scheme

Q10: Split

Implement `split-at`, which takes a list `lst` and a non-negative number `n` as input and returns a pair `new` such that `(car new)` is the first `n` elements of `lst` and `(cdr new)` is the remaining elements of `lst`. If `n` is greater than the length of `lst`, `(car new)` should be `lst` and `(cdr new)` should be `nil`.

```
scm> (car (split-at '(2 4 6 8 10) 3))
(2 4 6)
scm> (cdr (split-at '(2 4 6 8 10) 3))
(8 10)
```

```
(define (split-at lst n)
  (cond ((= n 0) (cons nil lst))
        ((null? lst) (cons lst nil))
        (else (let ((rec (split-at (cdr lst) (- n 1))))
                (cons (cons (car lst) (car rec)) (cdr rec)))))
)
```

Use Ok to test your code:

```
python3 ok -q split-at
```

Scheme Data Abstraction

Q11: Filter Odd Tree

Write a function `filter-odd` which takes a `tree` data abstraction and returns a new `tree` with all even `label`s replaced with `nil`.

> Consider using the `map` procedure to apply a one-argument function to a list.

Below is a Scheme-ified data abstraction of the Tree class we've been working with this semester.

```
; Constructs tree given label and list of branches
(tree label branches)

; Returns the label of the tree
(label t)

; Returns the list of branches of the given tree
(branches t)
```

```
(define (filter-odd t)
        (cond
                ((null? t) nil)
                ((odd? (label t)) (tree (label t) (map filter-odd (branches t))))
                (else (tree nil (map filter-odd (branches t))))
        )
)
```

Use Ok to test your code:

```
python3 ok -q filter_odd
```

Trees

Q12: Add trees

Define the function `add_trees`, which takes in two trees and returns a new tree where each corresponding node from the first tree is added with the node from the second tree. If a node at any particular position is present in one tree but not the other, it should be present in the new tree as well.

> **Hint:** You may want to use the built-in zip function to iterate over multiple sequences at once.

```
def add_trees(t1, t2):
    """
    >>> numbers = Tree(1,
    ...                [Tree(2,
    ...                      [Tree(3),
    ...                       Tree(4)]),
    ...                 Tree(5,
    ...                      [Tree(6,
    ...                            [Tree(7)]),
    ...                       Tree(8)])])
    >>> print(add_trees(numbers, numbers))
    2
      4
        6
        8
      10
        12
          14
        16
    >>> print(add_trees(Tree(2), Tree(3, [Tree(4), Tree(5)])))
    5
      4
      5
    >>> print(add_trees(Tree(2, [Tree(3)]), Tree(2, [Tree(3), Tree(4)])))
    4
      6
      4
    >>> print(add_trees(Tree(2, [Tree(3, [Tree(4), Tree(5)])]), \
    Tree(2, [Tree(3, [Tree(4)]), Tree(5)])))
    4
      6
        8
        5
      5
    """
    if not t1:
        return t2
    if not t2:
        return t1
    new_label = t1.label + t2.label
    t1_branches, t2_branches = list(t1.branches), list(t2.branches)
    length_t1, length_t2 = len(t1_branches), len(t2_branches)
    if length_t1 < length_t2:
        t1_branches += [None for _ in range(length_t1, length_t2)]
    elif length_t1 > length_t2:
        t2_branches += [None for _ in range(length_t2, length_t1)]
    return Tree(new_label, [add_trees(branch1, branch2) for branch1, branch2 in zip(t1_
```

Use Ok to test your code:

```
python3 ok -q add_trees
```

Regex

Q13: Address First Line

Write a regular expression that parses strings and returns whether it contains the first line of a US mailing address.

US mailing addresses typically contain a block number, which is a sequence of 3-5 digits, following by a street name. The street name can consist of multiple words but will always end with a street type abbreviation, which itself is a sequence of 2-5 English letters. The street name can also optionally start with a cardinal direction ("N", "E", "W", "S"). Everything should be properly capitalized.

Proper capitalization means that the first letter of each name is capitalized. It is fine to have things like "WeirdCApitalization" match.

See the doctests for some examples.

```
def address_oneline(text):
    """
    Finds and returns if there are expressions in text that represent the first line
    of a US mailing address.

    >>> address_oneline("110 Sproul Hall, Berkeley, CA 94720")
    True
    >>> address_oneline("What's at 39177 Farwell Dr? Is there a 39177 Nearwell Dr?")
    True
    >>> address_oneline("I just landed at 780 N McDonnell Rd, and I need to get to 1880
    True
    >>> address_oneline("123 Le Roy Ave")
    True
    >>> address_oneline("110 Unabbreviated Boulevard")
    False
    >>> address_oneline("790 lowercase St")
    False
    """
    block_number = r"\d{3,5}"
    cardinal_dir = r"([NEWS]\s)?" # whitespace is important!
    street = r"([A-Z][A-Za-z]+\s)+"
    type_abbr = r"[A-Z][a-z]{1,4}\b"
    street_name = f"{cardinal_dir}{street}{type_abbr}"
    return bool(re.search(f"{block_number} {street_name}", text))
```

Use Ok to test your code:

```
python3 ok -q address_oneline
```

Objects

Let's implement a game called Election. In this game, two players compete to try and earn the most votes. Both players start with 0 votes and 100 popularity.

The two players alternate turns, and the first player starts. Each turn, the current player chooses an action. There are two types of actions:

- The player can debate, and either gain or lose 50 popularity. If the player has popularity `p1` and the other player has popularity `p2`, then the probability that the player gains 50 popularity is `max(0.1, p1 / (p1 + p2))` Note that the `max` causes the probability to never be lower than 0.1.
- The player can give a speech. If the player has popularity `p1` and the other player has popularity `p2`, then the player gains `p1 // 10` votes and popularity and the other player loses `p2 // 10` popularity.

The game ends when a player reaches 50 votes, or after a total of 10 turns have been played (each player has taken 5 turns). Whoever has more votes at the end of the game is the winner!

Q14: Player

First, let's implement the `Player` class. Fill in the `debate` and `speech` methods, that take in another `Player other`, and implement the correct behavior as detailed above. Here are two additional things to keep in mind:

- In the `debate` method, you should call the provided `random` function, which returns a random float between 0 and 1. The player should gain 50 popularity if the random number is smaller than the probability described above, and lose 50 popularity otherwise.
- Neither players' popularity should ever become negative. If this happens, set it equal to 0 instead.

```
### Phase 1: The Player Class
class Player:
    """
    >>> random = make_test_random()
    >>> p1 = Player('Hill')
    >>> p2 = Player('Don')
    >>> p1.popularity
    100
    >>> p1.debate(p2)  # random() should return 0.0
    >>> p1.popularity
    150
    >>> p2.popularity
    100
    >>> p2.votes
    0
    >>> p2.speech(p1)
    >>> p2.votes
    10
    >>> p2.popularity
    110
    >>> p1.popularity
    135



    >>> # Additional correctness tests
    >>> p1.speech(p2)
    >>> p1.votes
    13
    >>> p1.popularity
    148
    >>> p2.votes
    10
    >>> p2.popularity
    99
    >>> for _ in range(4):  # 0.1, 0.2, 0.3, 0.4
    ...     p1.debate(p2)
    >>> p2.debate(p1)
    >>> p2.popularity
    49
    >>> p2.debate(p1)
    >>> p2.popularity
    0
    """
    def __init__(self, name):
        self.name = name
        self.votes = 0
        self.popularity = 100
```

```
    def debate(self, other):
        prob = max(0.1, self.popularity / (self.popularity + other.popularity))
        if random() < prob:
            self.popularity += 50
        else:
            self.popularity = max(0, self.popularity - 50)

    def speech(self, other):
        self.votes += self.popularity // 10
        self.popularity += self.popularity // 10
        other.popularity -= other.popularity // 10

    def choose(self, other):
        return self.speech
```

Use Ok to test your code:

```
python3 ok -q Player
```

Q15: Game

Now, implement the `Game` class. Fill in the `play` method, which should alternate between the two players, starting with `p1`, and have each player take one turn at a time. The `choose` method in the `Player` class returns the method, either `debate` or `speech`, that should be called to perform the action.

In addition, fill in the `winner` method, which should return the player with more votes, or `None` if the players are tied.

```
### Phase 2: The Game Class
class Game:
    """
    >>> p1, p2 = Player('Hill'), Player('Don')
    >>> g = Game(p1, p2)
    >>> winner = g.play()
    >>> p1 is winner
    True



    >>> # Additional correctness tests
    >>> winner is g.winner()
    True
    >>> g.turn
    10
    >>> p1.votes = p2.votes
    >>> print(g.winner())
    None
    """
    def __init__(self, player1, player2):
        self.p1 = player1
        self.p2 = player2
        self.turn = 0

    def play(self):
        while not self.game_over():
            if self.turn % 2 == 0:
                curr, other = self.p1, self.p2
            else:
                curr, other = self.p2, self.p1
            curr.choose(other)(other)
            self.turn += 1
        return self.winner()

    def game_over(self):
        return max(self.p1.votes, self.p2.votes) >= 50 or self.turn >= 10

    def winner(self):
        if self.p1.votes > self.p2.votes:
            return self.p1
        elif self.p2.votes > self.p1.votes:
            return self.p2
        else:
            return None
```

Use Ok to test your code:

```
python3 ok -q Game
```

Q16: New Players

The `choose` method in the `Player` class is boring, because it always returns the `speech` method. Let's implement two new classes that inherit from `Player`, but have more interesting `choose` methods.

Implement the `choose` method in the `AggressivePlayer` class, which returns the `debate` method if the player's popularity is less than or equal to `other`'s popularity, and `speech` otherwise. Also implement the `choose` method in the `CautiousPlayer` class, which returns the `debate` method if the player's popularity is 0, and `speech` otherwise.

```
### Phase 3: New Players
class AggressivePlayer(Player):
    """
    >>> random = make_test_random()
    >>> p1, p2 = AggressivePlayer('Don'), Player('Hill')
    >>> g = Game(p1, p2)
    >>> winner = g.play()
    >>> p1 is winner
    True



    >>> # Additional correctness tests
    >>> p1.popularity = p2.popularity
    >>> p1.choose(p2) == p1.debate
    True
    >>> p1.popularity += 1
    >>> p1.choose(p2) == p1.debate
    False
    >>> p2.choose(p1) == p2.speech
    True
    """
    def choose(self, other):
        if self.popularity <= other.popularity:
            return self.debate
        else:
            return self.speech

class CautiousPlayer(Player):
    """
    >>> random = make_test_random()
    >>> p1, p2 = CautiousPlayer('Hill'), AggressivePlayer('Don')
    >>> p1.popularity = 0
    >>> p1.choose(p2) == p1.debate
    True
    >>> p1.popularity = 1
    >>> p1.choose(p2) == p1.debate
    False



    >>> # Additional correctness tests
    >>> p2.choose(p1) == p2.speech
    True
    """
    def choose(self, other):
        if self.popularity == 0:
            return self.debate
        else:
            return self.speech
```

Use Ok to test your code:

```
python3 ok -q AggressivePlayer
python3 ok -q CautiousPlayer
```

Lists

Q17: Intersection - Summer 2015 MT1 Q4

Implement `intersection(lst_of_lsts)`, which takes a list of lists and returns a list of distinct elements that appear in all the lists in `lst_of_lsts`. If no number appears in all of the lists, return the empty list. You may assume that `lst_of_lsts` contains at least one list.

```
def intersection(lst_of_lsts):
    """Returns a list of distinct elements that appear in every list in
    lst_of_lsts.

    >>> lsts1 = [[1, 2, 3], [1, 3, 5]]
    >>> intersection(lsts1)
    [1, 3]
    >>> lsts2 = [[1, 4, 2, 6], [7, 2, 4], [4, 4]]
    >>> intersection(lsts2)
    [4]
    >>> lsts3 = [[1, 2, 3], [4, 5], [7, 8, 9, 10]]
    >>> intersection(lsts3)         # No number appears in all lists
    []
    >>> lsts4 = [[3, 3], [1, 2, 3, 3], [3, 4, 3, 5]]
    >>> intersection(lsts4)         # Return list of distinct elements
    [3]
    """
    elements = []
    for elem in lst_of_lsts[0]:
        condition = elem not in elements
        for lst in lst_of_lsts[1:]:
            if elem not in lst:
                condition = False
        if condition:
            elements = elements + [elem]
    return elements
```

Use Ok to test your code:

```
python3 ok -q intersection
```

Q18: Deck of cards

Write a list comprehension that will create a deck of cards, given a list of `suits` and a list of `ranks`. Each element in the list will be a card, which is represented by a 2-element list of the form `[suit, rank]`.

```
def deck(suits, ranks):
    """Creates a deck of cards (a list of 2-element lists) with the given
    suits and ranks. Each element in the returned list should be of the form
    [suit, rank].

    >>> deck(['S', 'C'], [1, 2, 3])
    [['S', 1], ['S', 2], ['S', 3], ['C', 1], ['C', 2], ['C', 3]]
    >>> deck(['S', 'C'], [3, 2, 1])
    [['S', 3], ['S', 2], ['S', 1], ['C', 3], ['C', 2], ['C', 1]]
    >>> deck([], [3, 2, 1])
    []
    >>> deck(['S', 'C'], [])
    []
    """
    return [[suit, rank] for suit in suits
                         for rank in ranks]
```

Use Ok to test your code:

```
python3 ok -q deck
```